1.

I noticed that this is similar to the Ruby example where we used the comparison operator on the strings and they still yielded true. This was different from any other language since typically in Java/Python the object_ids are usually different for all types.

2.

a. Pass by object reference because the values can still be mutated

b. It would likely be pass by value, because then the variables wouldn't be revised.

c. It would return 2 if it was passed by value and 5 if it was pass by object reference. This is because the value can't be revised if it is passed to a function parameter in pass by value but it can if it was pass by reference.

3.

a.

```
"bar"
"baz"
1
```

b. we aren't passing in a reference so error?

c.

```
"bar"
"baz"
4
```

d.

```
"bar"
"baz"
4
```

4.

For the optional struct, we explicitly communicate a potential failure due to our absence of a dedicated type value. Thus, the API would need to explicitly check the absense of such value based on a more manual approach handling such a case. Meanwhile for the exception handling approach, we rely more on catching and throwing exceptions which provides a distinct separation between the normal and exceptional control flow. Additionally, we would need to keep track of the potential exceptions and implement try-catch blocks to catch certain errors. Ultimatley, the optional T struct is more desirable for the non-dedicate type value actually has an expected value, while the exception handling approach would be more desirable for handling exceptional/error situations.

5.

a. catch 1, hurray!, I'm done!

b. catch 1, hurray!, I'm done!

c. Output: catch 1, hurray!, I'm done!, that's what I say

d. a bad_exception will be thrown in foo(3)

e. hurray!, I'm done!, that's what I say